ML 03: LOGISTIC REGRESSION

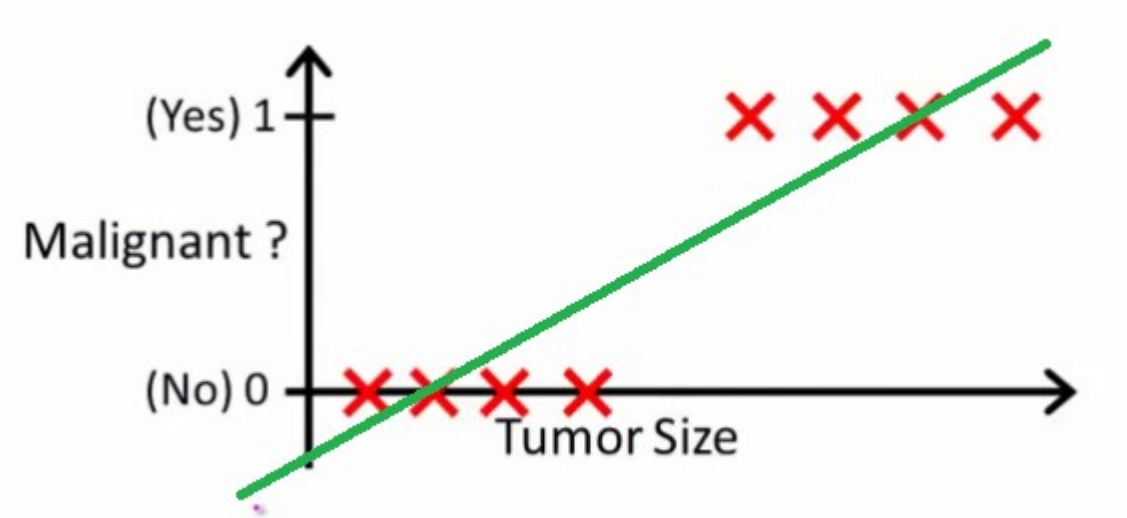
지난 시간엔 *Regression* 을 해결하기 위해 *graident descent* 알고리즘을 도입했었다. *learning rate*, *vectorization* 등에 대해서 알아 보기도 했고. 이번시간엔 *classification* 과 *regulrzation* 에 대해서 배워 본다.

이 수업이 재밌는 이유는 수식을 증명하는 것보다 수식속에 숨겨진 내용들을 직관적으로 이해할 수 있게 설명하기 때문이다. ~~그러나 교수님 과제는 제발 그만~~

Classification

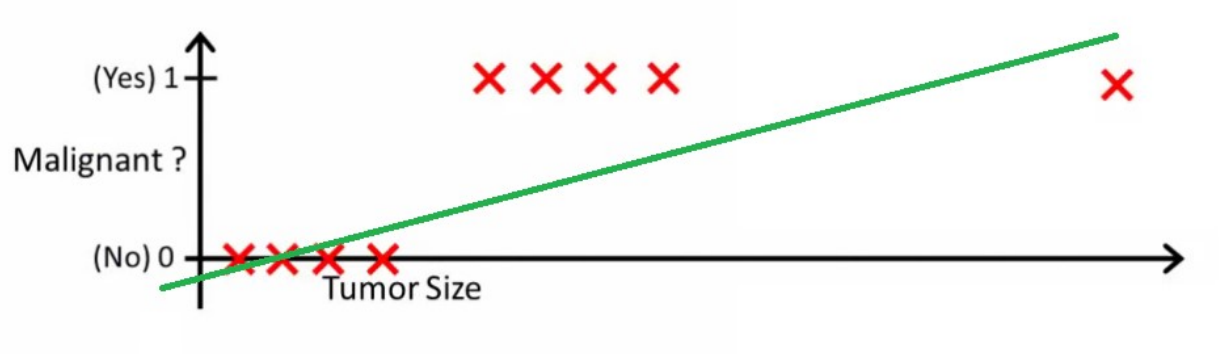
regression 이 *continuous value* 를 다룬다면 **Classification** 은 **discrete value** 를 다룬다. 따라서 *Classification (분류)* 의 예는,

- 이메일이 스팸인지 / 아닌지
- 온라인 거래가 사기인지 / 아닌지 (Online Transaction: Fraudulent)
- 악성 종양인지 / 아닌지



(http://stats.stackexchange.com)

위와 같은 경우, *Regression* 으로 문제를 풀면 당장은 맞아 보이나, 종양이 이상한 위치에 생겼을 경우 아래와 같이 직선이 크게 변한다.



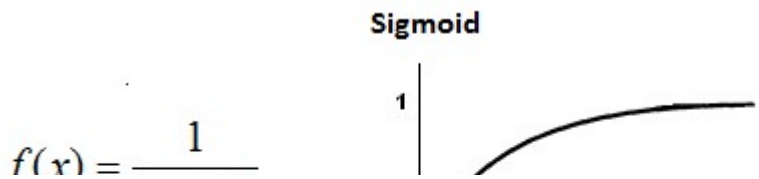
(http://stats.stackexchange.com)

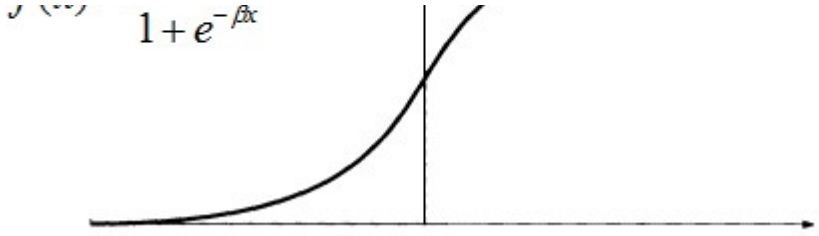
따라서 이렇게 *discrete value* 에 대해서는 *Regression* 보다는 *Threshold* 에 기반을 두어, `h(x)` 가 일정 값 이상이면 `y=1` 로 예측하는 편이 더 정확도가 높아진다. 게다가 *regression* 은 직선이기 때문에, `0 <= y <= 1` 인 `y` 에 대해서 0보다 작거나, 1보다 더 큰 `y` 를 만들어낼 수 있다.

이런 이유 때문에 *Classification* 문제에 *Regression* 을 잘 사용하지 않는다. 그러나 `y` 의 범위가 `0 <= h(x) <= 1` 을 가지는 *Logistic Regression* 도 있다. 이건 *Classification* 에 사용되기도 한다.

Logistic Regression

이전에 언급했듯이 *classification* 에선 예측된 값, 즉 `h(x)` 값이 0 과 1사이에 있길 바란다. 이를 위해 *logistic function*, 혹은 **sigmoid function** 이라 불리는 아래 식을 *hypothesis* `h(x)` 에 적용하면 아래와 같은 그림이 나온다.





(http://www.saedsayad.com)

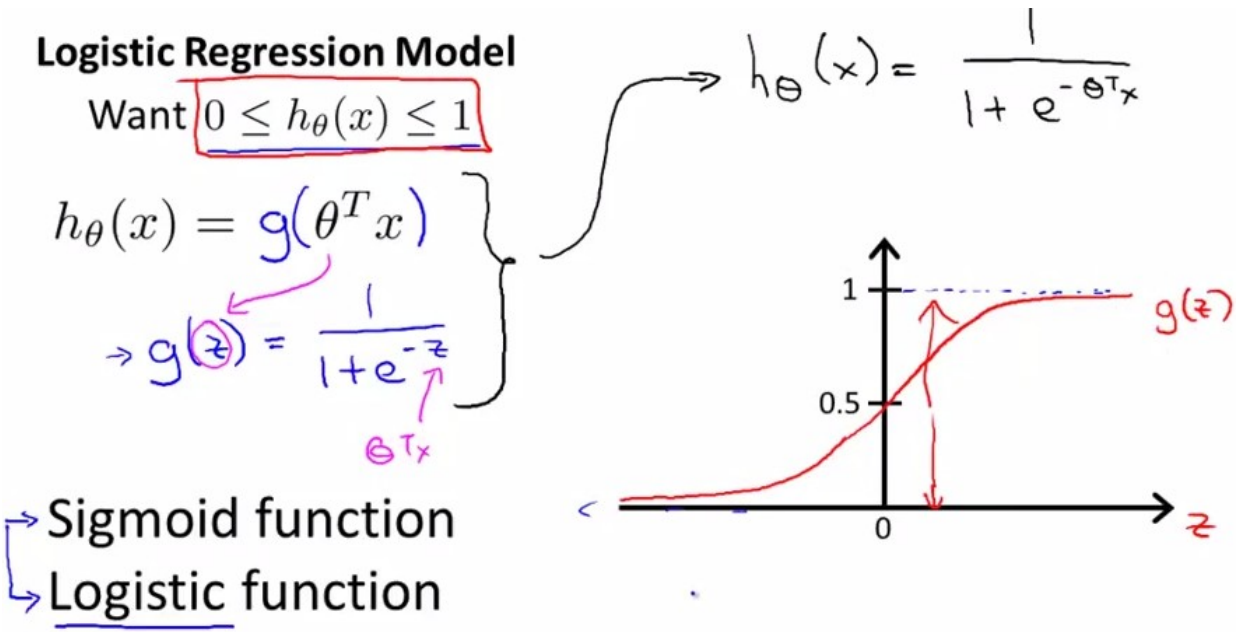
이 때 *sigmoid function* 이 적용된 `h(x)` 는 최대값이 1이므로, 이건 입력값 `x` 에 대해서 `y` 가 1이 나올 확률이라 보아도 된다. 따라서

```
h(x) = P(y = 1 | x ; 0)
```

> Probability that `y = 1`, given `x`, parameterized by `0(theta)`

이 때 *sigmoid function* 을 보면, X 축이 0보다 큰 점에선 `y` 값이 0.5 보다 크므로, 이 점 이후부터는 `y` 를 1 이라 *예측 (predict)* 하고, 반대로 X 축 값이 0보다 작은 지점에선 `y` 를 0이라 예측할 수 있다.

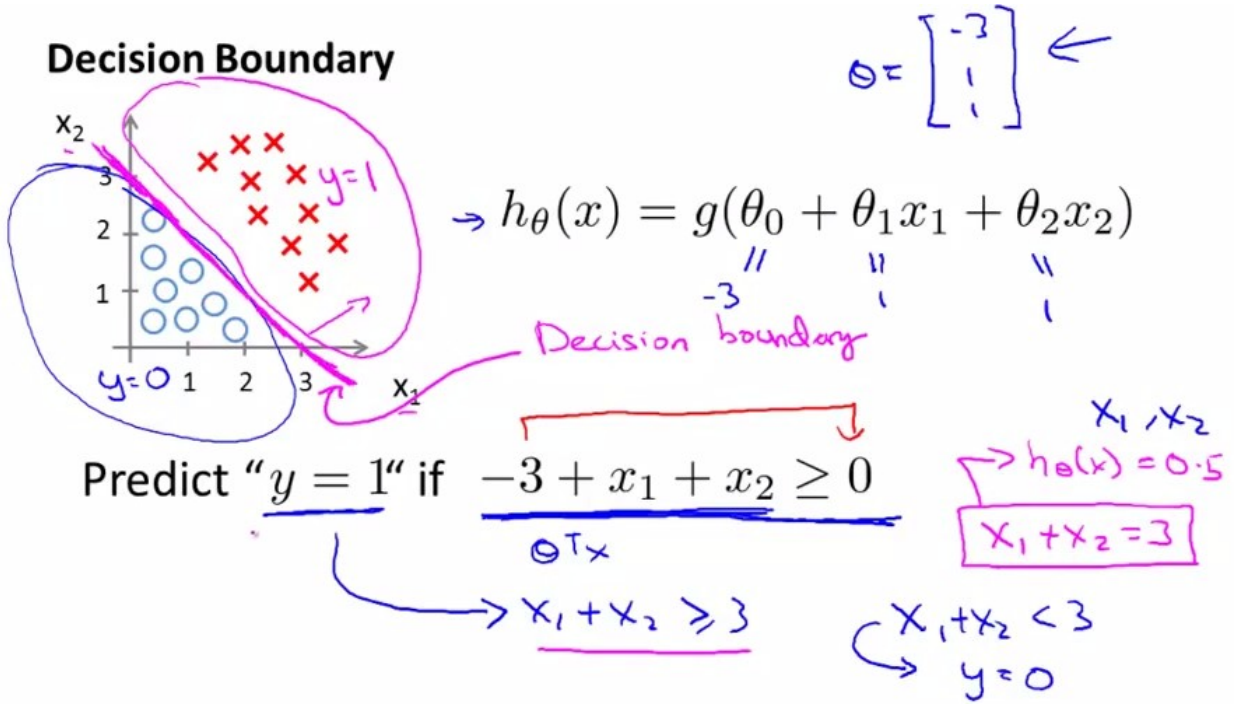
그런데 `h(x) = g(0^T * x)` 이므로, 본래의 *hypothesis* `0^T * x` 가 0이 되는 지점을 찾으면 된다.



(http://blog.csdn.net/abcjennifer/)

Decision Boundary

이제 실제로 문제에 적용해 보자. 다음과 같이 두개의 집단이 있을때, 이 두 집단을 가르는 식을 찾기 위한 `h(x) = g(01 + 01x1 + 02x2)` 가 있다고 해 보자.

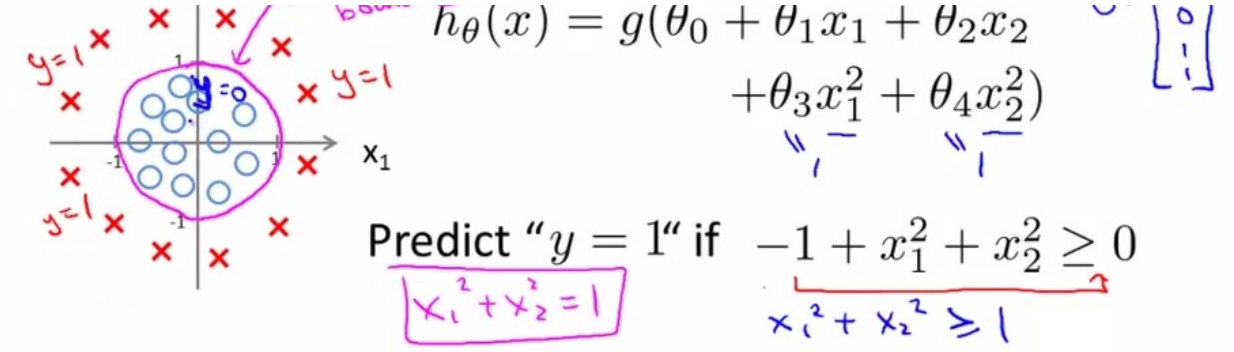


(http://blog.csdn.net/abcjennifer/)

이때 `0(theta)` 를 `[-3; 1; 1]` 로 잡으면 `y` 가 `1` 이 되는 지점은 `0^T * x >= 0` 인 지점, 즉 `-3 + x1 + x2 >= 0` 인지점을 찾으면 된다. 이 식을 풀어서 쓰면

`x1 + x2 => 3` 이므로, 위 그림에서 분홍색 선을 찾을 수 있다. 이 선을 **Decision Boundary** 라 부른다. 그리고 이 *Decision Boundary* 는 `g(z) = 0` 즉, `h(x) = 0.5` 인 지점이다.

Non-linear dicision boundary 는 어떨까?



(http://blog.csdn.net/abcjennifer/)

이 경우 `x1^2`, `x2^2` 이라는 새로운 *feature* 를 도입하고, *parameter* 인 `theta` 를 `[-1; 0; 0; 1; 1;]` 로 잡았다. 식을 풀면, 위와 같은 원 형태의 *Decision Boundary* 가 나온다.

feature 만 잘 조합하면, 즉 *polynomial* 만 잘 만들면 땅콩이나 하트모양 등의 *Decision boundary* 도 만들 수 있다.

Cost Function

이제 문제는 `theta` 를 어떻게 고르느냐 하는건데, 식을 좀 다시 살펴보자.

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \frac{1}{2} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$$

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \text{Cost}(h_\theta(x), y)$$

(http://www.holehouse.org/)

Linear regression 에서 사용하는 *cost function* 에 지금의 `h(x)`, 즉 *sigmoid function* 이 적용된 `h(x)` 를 제곱한 `J(0)` 는 *non-convex* 형태가 된다. 따라서 *global optimum* 보다는 *local optimum* 을 찾게 된다.

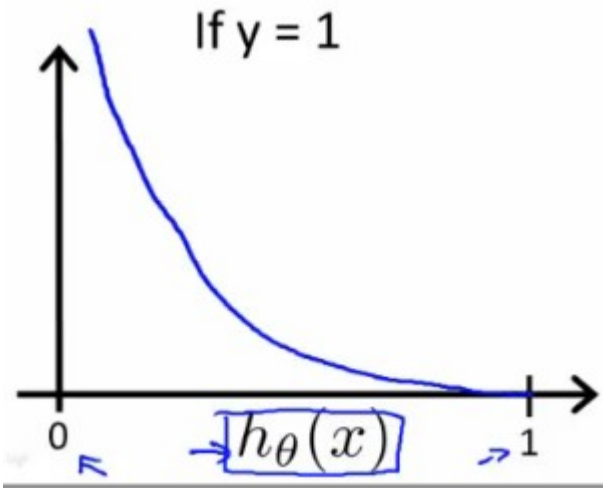
이를 방지하기 위해서, *convex* 형태의 *cost function* 을 사용해야 하는데,

$$\text{Cost}(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{if } y = 1 \\ -\log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$

(http://www.holehouse.org/)

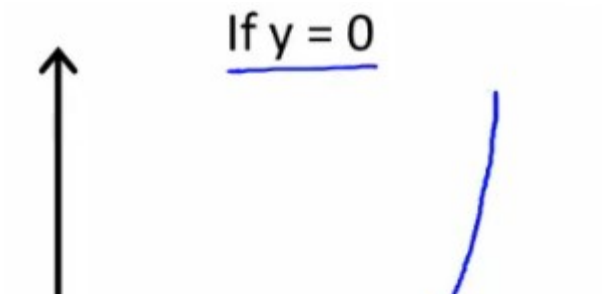
이 *cost function* 을 사용하면, `y = 1` 일때 다음과 같은 그래프를 얻게 된다. `0 <= h(x) <= 1` 임을 참고하자. `y = 1` 일때, `h(x) = 0` 으로 가면, *cost function* 의 값, 즉 *cost* 자체가 높아지므로, *Cost* 를 낮추는 반대 방향으로 움직이게 된다.

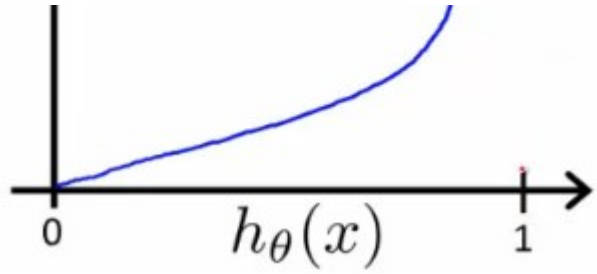
직관적으로 보면, `h(x)` 자체는 `y = 1` 일 확률인데, `y = 1` 일때, `h(x) = 0` 이라는 것은 말이 안 되므로 비용이 무한대로 증가하는 것이 말이 된다.



(http://www.holehouse.org/)

반대로 `y = 0` 일때의 그래프를 보면 `h(x) = 0` 즉, `y = 0` 일 확률이 `0` 으로 갈때 *cost* 가 감소한다.





(http://www.holehouse.org/)

결국 아래의 새로운 *logistic regression cost function* 을 이용하면, `J(0)` 를 *convex function* 으로 만들 수 있다.

$$\text{Cost}(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{if } y = 1 \\ -\log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$

(http://www.holehouse.org/)

Simplified Cost Function and Gradient Descent

이제 `y = 0`, `y = 1` 로 나누어져 있던 *cost function* 을 좀 더 간단히 표현해 보자.

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \text{Cost}(h_\theta(x^{(i)}), y^{(i)})$$

$$\text{Cost}(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{if } y = 1 \\ -\log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$

Note: $y = 0$ or 1 always

(http://www.holehouse.org/)

다음과 동일하다. `y = 0`, `y = 1` 을 직접 넣어보면 금방 알 수 있다.

```
cost(hθ(x),y) = -y * log(hθ(x)) - (1-y) * log(1 - hθ(x))
```

자 이제 다시 본론으로 돌아와서, 우리는 처음에 `theta` 를 찾길 원했고, 그래서 *gradient descent* 를 쓰려고 했는데, 마침 보니 `h(x)` 가 *sigmoid function* 이 적용된 형태라서 *non-convex function* 이므로, `h(x)` 를 포함한 *cost-function* 이 *convex function* 이 되는 식을 찾아냈다. 이제 그 식을 *gradient descent* 에 적용하면,

$$J(\theta) = -\frac{1}{m}[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log (1 - h_\theta(x^{(i)}))]$$

이고, 이제 이걸 *batch gradient descent* 에 적용하면 아래와 같은데, 여기에 *partial derivative* 를 적용하면

$$\text{Repeat } \{ \theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta) \}$$

놀랍게도 *linear regression* 과 같은 식이 나온다. 오오-머신러닝-오오

Repeat {

$$\theta_j := \theta_j - \alpha \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

(simultaneously update all θ_j)

}

(http://www.holehouse.org/)

다만 다른점은 *hypothesis* 가 *sigmoid function* 을 적용한 형태라는 것,

$$h_\theta(x) = \frac{1}{}$$

$$h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$$

(http://www.holehouse.org/)

Advanced Optimization

위에서 보았겠지만, `J(0)` 의 최소값을 찾기 위해서는 아래 두개의 값을 구해야 한다.

$$J(\theta)$$
$$\frac{\partial}{\partial \theta_j} J(\theta) \quad (\text{for } j = 0, 1, \ldots, n)$$

(http://www.holehouse.org/)

이 값들을 이용해서 *gradient descent* 대신 다음의 알고리즘을 사용할 수 있다.

(1) Conjugate gradient
(2) BFGS
(3) L-BFGS

이 알고리즘들의 장점은, *leanring rate* 를 고를 필요가 없고, 대부분 *gradient decsent* 보다 빠르다.

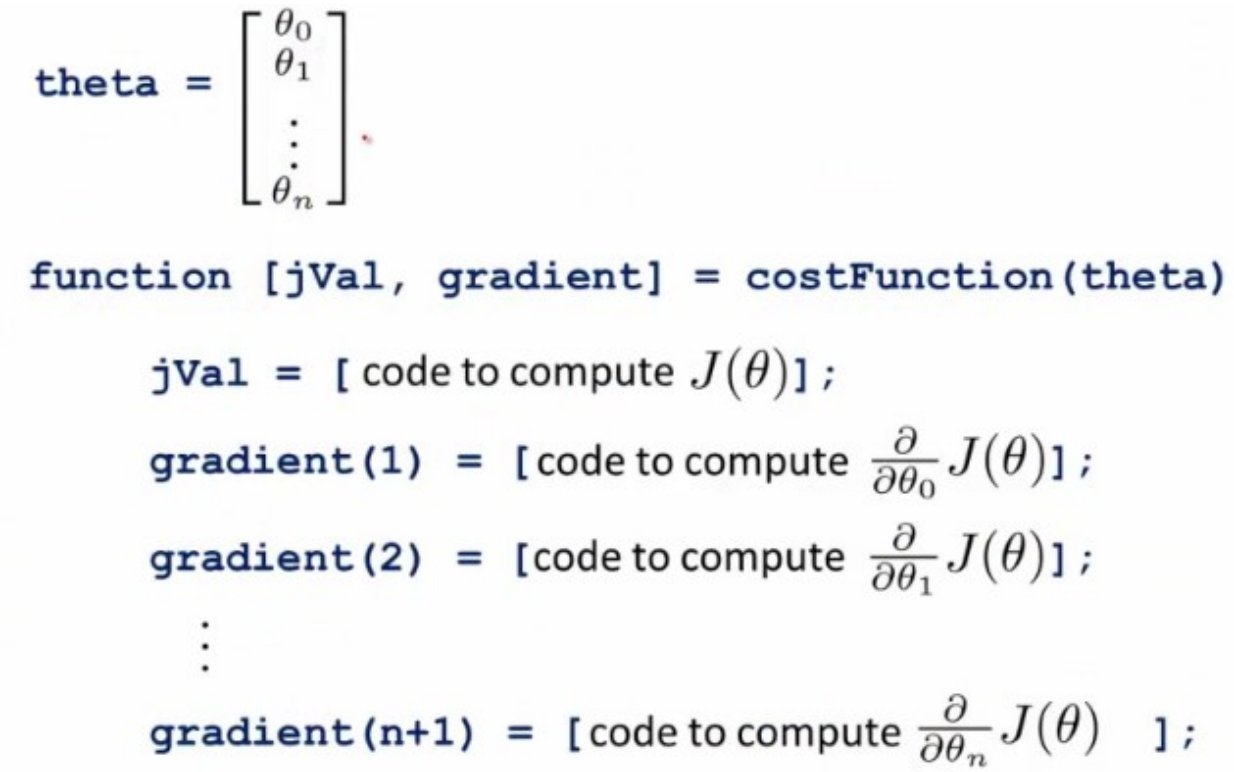
그러나 더 복잡하고, 라이브러리마다 구현이 다를 수 있으며, 디버깅이 힘들수 있다. 자 이제 *advanced optimization* 을 이용해 보자.

Example:

$$\theta = \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix}$$
$$J(\theta) = (\theta_1 - 5)^2 + (\theta_2 - 5)^2$$
$$\frac{\partial}{\partial \theta_1} J(\theta) = 2(\theta_1 - 5)$$
$$\frac{\partial}{\partial \theta_2} J(\theta) = 2(\theta_2 - 5)$$

(http://www.holehouse.org/)

위와 같은 식에 대해서 *cost function* 을 `octave` 에서 이렇게 만들 수 있다.



(http://www.holehouse.org/)

이제, `octave` 에서 제공해 주는 `fminunc` 에 우리가 만든 `costFunction` 과 초기 `theta` 값, 그리고 옵션을 집어 넣으면

```
% define the options data structure
options= optimset('GradObj', 'on', 'MaxIter', '100');

% set the initial dimensions for theta % initialize the theta values
initialTheta= zeros(2,1);

% run the algorithm
[optTheta, funtionVal, exitFlag]= fminunc(@costFunction,
initialTheta, options);
```

`optTheta` 는 우리 찾길 원했던 `theta` 값이고, `functionVal` 은 최종 *cost* 를 돌려준다. `exstFlag` 는 알고리즘이 수렴했는지, 아닌지 알려준다.

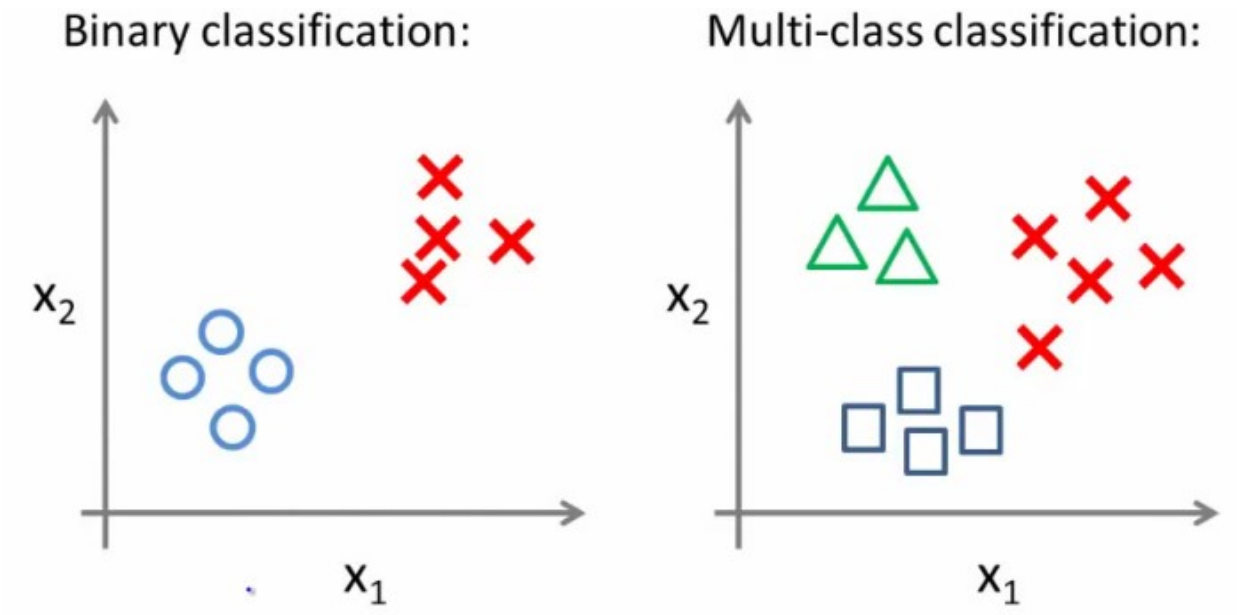
만약 *logistic regression* 에 대한 `theta` 값을 찾고 싶으면, *cost function* 을 *logistic regression* 에 맞게 작성하면 된다.

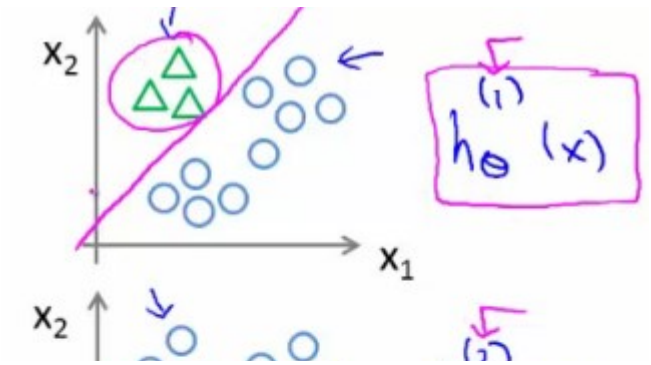
Multiclass Classification

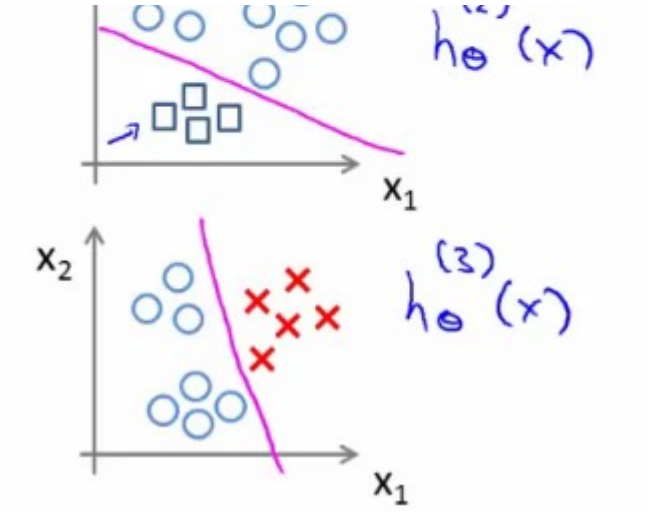
이제 단순히 `y = 0 or 1` (*binary classification*) 이 아닌, 다양한 *class* 가 있는 *classification* 을 고려해보자, 예를 들면 날씨는 `sunny`, `cloudy` , `hot`, `cold` 등으로 분류될 수 있다.

one-vs-all (One-vs-rest)

multi class 를 분류할 수 있는 한가지 방법은, 하나를 정하고, 그 나머지와 분류하는것이다. 이걸 *class* 갯수만큼 진행하면,







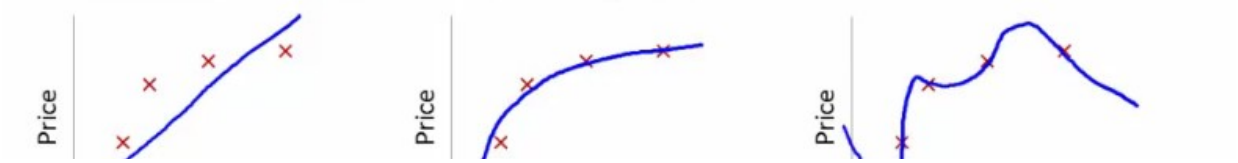
(http://www.holehouse.org/)

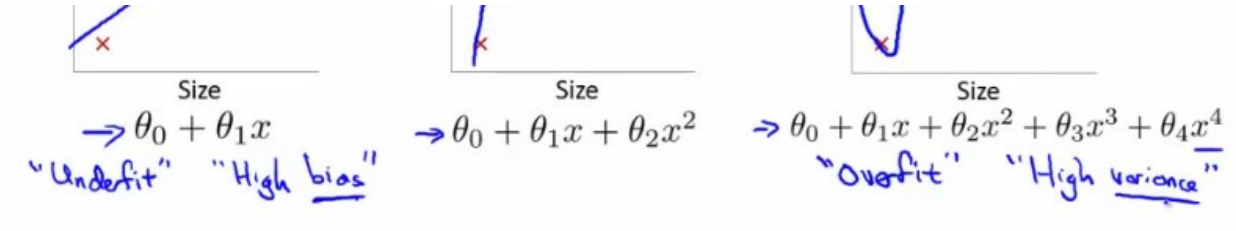
위 그림과 같은 경우, *class* 가 3개기 때문에 `(i = 1, 2, 3)` 으로 놓으면 `i` 마다 각각의 `hθ^(i)(x)` 값, 즉 예측 값을 얻을 수 있다. 따라서 새로운 무언가가 input 으로 들어왔을때, `hθ^(i)(x)` 값을 최대로 해주는 `i` 을 선택하면 분류가 된다. ~~참-쉽죠?~~

Overfitting

Overfitting 은 너무나 많은 *feature* 가 있을 때는 *cost function* 이 트레이닝 셋에 잘 맞아 `0` 에 수렴 하지만, 새로운 데이터가 들어왔을때는 예측을 잘 하지 못하는 경우를 말한다. 다시 말해 *hypothesis* 가 너무 고차원의 다항식이어서 그렇다. *(too many parameters)* 즉 아래 그림에서 좌측은 경향을 나타내긴 하지만 모든 트레이닝셋을 경유하는 직선은 만들어내지 못했다. (*under fit*) 반면 가장 우측은, 트레이닝셋을 모두 경유하는 *hypothesis* 를 만들어 냈지만, 다항식의 차수가 너무 높아 새로운 데이터가 들어왔을 때 예측하지 못할 수가 있다. *can't apply, unable to generalize* 교수님은 다음과 같이 슬라이드에 적으셨다.

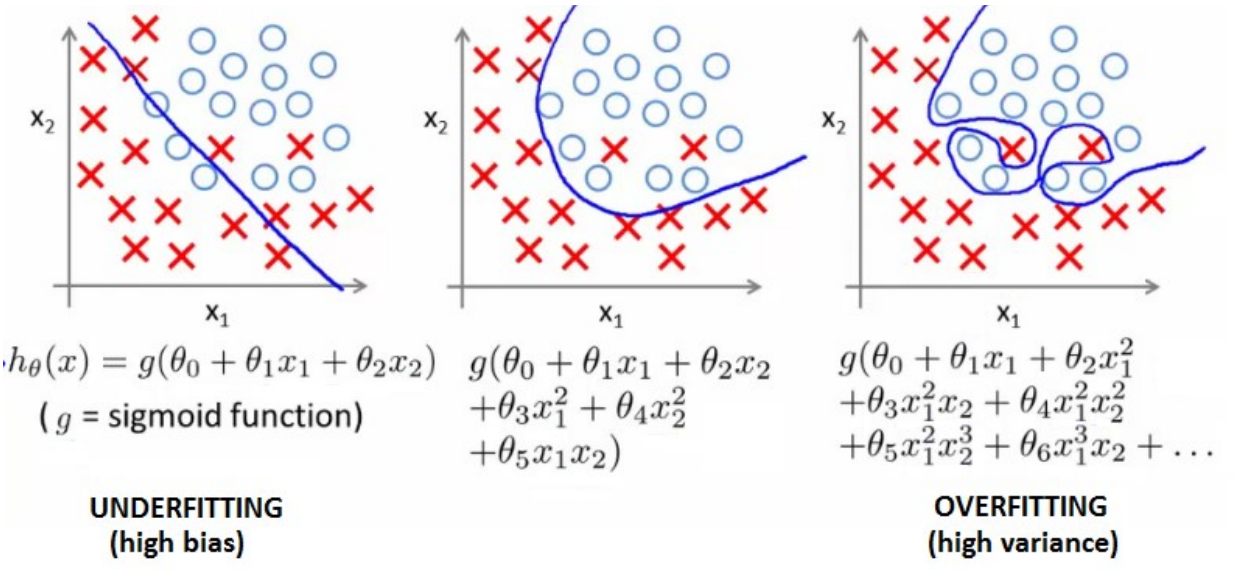
> It makes accurate predictions for examples in the training set, but it does not generalize well to make accurate prediction on new, previously unseen examples





(http://www.holehouse.org/)

logistic regression 에서도 *Overfitting* 이 발생할 수 있다.



(http://www.holehouse.org/)

주로 *training set* 이 부족하고 *feature* 가 많을때 발생하는데 해결책은

(1) *feature* 를 줄일 수 있다. 수동으로 사용할 *feature* 를 선택하는 방법과 *Model selection algorithm* 을 사용할 수도 있다.
(2) *regularization* 을 이용한다. 모든 *feature* 를 유지하지만, 얼마나 각 *feature* 가 *prediction* 에 기여할지를 변경한다.

Regularization, Cost function

Regularization 은 원하는 파라미터가 *hypothesis* 에 기여하는 바를 조절하는 것이다. 우리가 만약에 `0_3` 과 `0_4` 를 최소화 하고 싶다고 하자. 그럼 다음과 같은 식을 만들면

된다. 전체 식의 최소값을 찾는 것이기 때문에, 상수가 `1000` 인 `θ_3`, `θ_4` 는 *0(zero)* 에 가까운 수가 나온다. 다시 말해서 이들 두 파라미터가 기여하는 바를 줄인 것이다.

$$\min_\theta \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + 1000\,\theta_3^2 + 1000\,\theta_4^2$$

(http://www.holehouse.org/)

parameters 가 작은 값을 가질수록 간단한 *hypothesis* 가 나오고, *overfitting* 하지 않는다. 이를 위해 `λ` 라는 *regularization parameter* 를 가진 식을 *cost function* 에 더 붙여 *parameter* 가 기여하는 바를 조절하면, 아래와 같은 식을 구할 수 있다. 참고로 뒷부분의 식은 *regularization term* 이라 부르는데, `j` 가 1부터 시작하는 것에 주목하자. 이는 `θ_0` 은 *regularization* 하지 않는다는 의미이다.

$$J(\theta) = \frac{1}{2m}\left[\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda\sum_{j=1}^{n}\theta_j^2\right]$$

$$\min_\theta J(\theta)$$

(http://www.holehouse.org/)

`λ` 가 매우 크면 어떻게 될까? `θ_0` 이외의 다른 파라미터는 0에 수렴 하므로, *hypothesis* 는 상수가 되어 트레이닝 셋에 *under fit* 할 것이다.

Regularized Linear Regression

regularization term 으 `j` 가 1부터 시작하므로, *cost function* 을 쉽게 계산하기 위해 분리하면 *gradient descent* 식은 다음과 같이 적을 수 있다.

Repeat {

$$\theta_0 := \theta_0 - \alpha\frac{1}{m}\sum_{}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_0^{(i)}$$

$$\theta_j := \theta_j - \alpha \quad \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)} \quad (j = 0, 1, 2, 3, \ldots, n) \}$$

(http://www.holehouse.org/)

이제 위 두 식에서 아래 식을 정리하면, 다음과 같고

$$\theta_j := \theta_j (1 - \alpha \frac{\lambda}{m}) - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

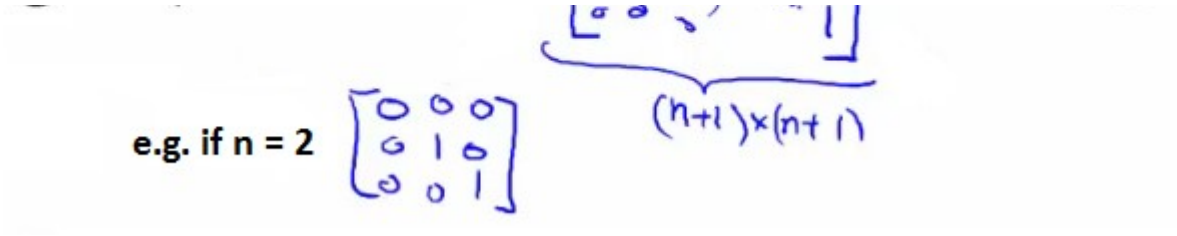
<면 된다. 이때 이 매트릭스의 `(0, 0)` 위 식에서 앞부분은 아래와 같다. 보통 `m` 이 매우 크고, `a` 가 매우 작으므로 위 값은 1보다 작다. 예를 들면 `0.99 * 0_j` 처럼.

$$(1 - \alpha \frac{\lambda}{m})$$

이제 *Normal equation* 에 어떻게 적용할지 고려해 보자, 본래 *normal equation* 식은 아래와 같은데,

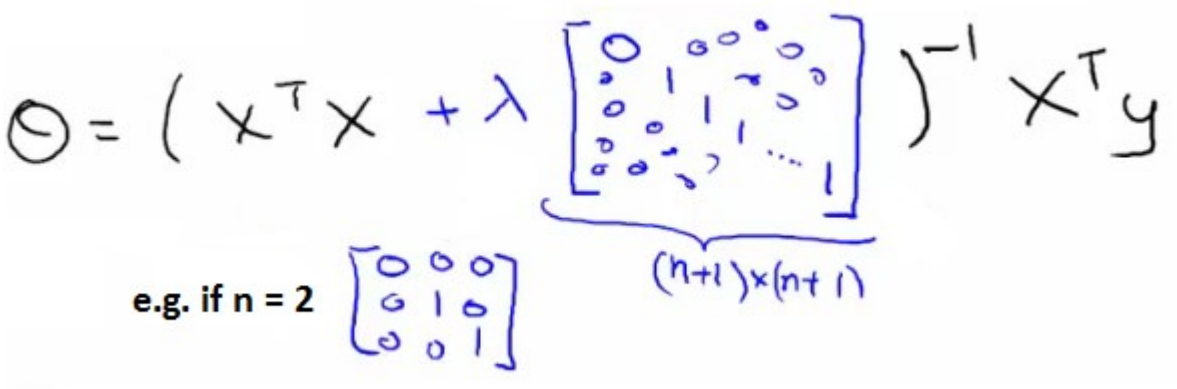
$$\theta = (X^T X)^{-1} X^T y$$

`X^T * X` 부분에 `λ` 가 곱해지는 `n+1 * n+1` 의 *matrix* 를 곱하면 된다. 이때 이 매트릭스의 `(0, 0)` 부분이 `0` 인 것은 `0_0` 에 *regularization* 을 적용하지 않기 위한 것.



그럼 만약에 `X^T * X` 가 *non-invertible* 이라면 어떻게 될까? 이건 지난 시간에 언급했듯이 *redundant feature* 가 너무 많거나, `m <= n` , 즉 트레이닝 셋에 비해 *feature* 가 너무 많을 때 발생한다고 말했다.

놀랍게도, `λ > 0` 이면, 아래 식에서 `X^T * X + λ` (λ's (0, 0) = 0) 은 제대로 *invertible* 함을 증명할 수 있다. 다시 말해서 *regularzation* 을 통해서 *non-invertible* 문제도 해결할 수 있다는 것.

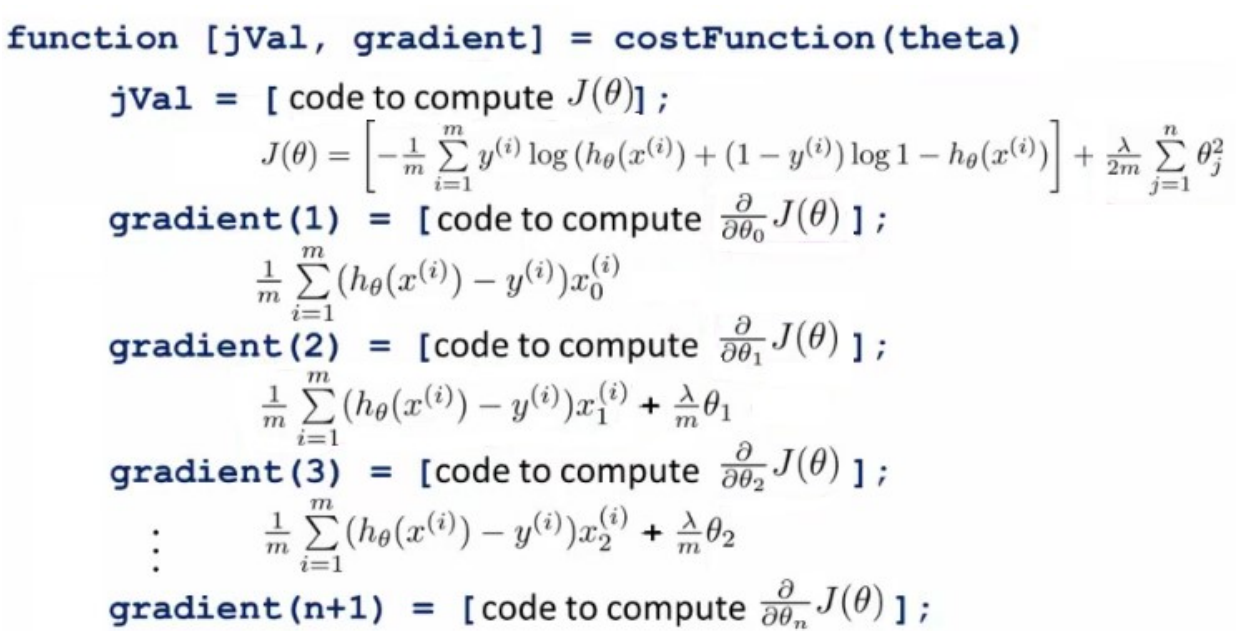


(http://www.holehouse.org/)

Regularized Logistic Regression

linear regression 과 마찬가지로 `θ(theta)` 를 0과 1로 분리해 *regularization term* 을 추가하면 된다. 다른점은 `h(x)` 가 *sigmoid function* 의 형태라는 것.

그리고 *gradient descent* 를 풀기 위해 *octave* 에서 제공하는 알고리즘들을(*conjugate*, *BFGS*, *L-BFGS* 등) 을 `fminunc` 이용해서 사용할 수 있다. 이를 위해 언급 했듯이 `jval` 과 `θ(theta)` 에 대한 `graident` 를 돌려주는 *cost function* 을 만들어야 하는데, *regularzation term* 이 추가되었으므로 해당하는 값을 더해서 각 `θ` 에 대한 *gradient* 를 계산하는 식을 만들어주면 된다.



(http://www.holehouse.org/)

Summary

3주째에는 *Classification* 과 *Regularization* 에 대해서 배웠다. 수업은 어렵지 않다. 과제가 문제지 ㅠㅠ 교수님. 파이썬으로 과제를 내주셨으면 좀 더 배우는 맛이 있었을텐데요!

References

(1) why-not-approach-classification-through-regression
(2) http://www.saedsayad.com
(3) http://blog.csdn.net/abcjennifer/
(4) http://www.holehouse.org/

Machine Learning by Andrew Ng, *Coursera*